Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

«Симуляция движения сфер под действием гравитационного поля»

Студент: Куликов Егор Андреевич

Группа: ИУ7-56Б

Руководитель: Клорикьян Петрос Вазгенович

Цель и задачи

Целью курсовой работы является реализация программы для моделирования движения группы сферических тел в пространстве под действием их собственного гравитационного поля.

Задачи:

- Провести анализ существующих алгоритмов компьютерной графики
- Выбрать подходящие для данной работы алгоритмы
- Выбрать язык программирования и среду разработки
- Реализовать выбранные алгоритмы в виде одной программы с графическим интерфейсом
- Провести исследование быстродействия разработанного ПО в зависимости от количества объектов сцены

Общий алгоритм решения задачи

Задача симуляции движения тел будет решаться как серия подзадач с временным интервалом δt между ними — состояний сцены в моменты времени кратные δt.
Для каждого состояния сцены:

- Рассчитать суммарную гравитационную силу, действующую на каждое тело со стороны других тел
- Используя вычисленные силы рассчитать ускорения, скорости и перемещения каждого тела за δt
- Определить, произошли ли столкновения тел, соединить столкнувшиеся тела с расчетом параметров (массы, скорости, ускорения) нового тела.
- Отрисовать полученное состояние сцены

Способ задания трехмерного объекта

Каркасная модель
- Совокупность вершин и ребер
- Нет информации о гранях
- Невозможно однозначно интерпретировать видимость или невидимость грани
- Невозможно применить модель освещения

Поверхностная модель
- **Предполагается что объекты ограничены оболочками**
- **Объекты имеют внутреннюю и внешнюю части**
- **Эти оболочки изображаются**
- **Любую поверхность можно аппроксимировать многогранниками**

Твердотельная модель
- Отличается от поверхностной тем, что хранит нормали
- Хранит информацию о том, с какой стороны от оболочки находится материал модели

Удаление невидимых линий и поверхностей

Алгоритм Робертса

- Работает в объектном пространстве
- Очень точный
- Имеет квадратичную сложность
- Вычисляет все пересечения объектов аналитически и точно (в данном случае избыточно)

Алгоритм Z-буфера

- **Использует две матрицы – буфер кадра, хранящий интенсивности пикселей, буфер глубины, хранящий глубину пикселей**
- **В процессе работы в буфер кадра заносится ближайший к наблюдателю пиксель, что и является процессом удаления невидимых линий и поверхностей**
- **Не более чем линейная сложность**
- **Произвольная сложность сцены**

Алгоритм обратной трассировки лучей

- Отслеживается траектория каждого луча из фокуса камеры в пиксель на экране
- Ищутся точки пересечения каждого луча со всеми объектами (затратно и долго, хоть и оптимизируется с помощью объемлющих оболочек)

Алгоритмы закраски

Однотонная

- Все полигоны закрашиваются одним цветом
- Видны границы полигонов, нереалистично
- Для каждого полигона считается всего одна нормаль

По Гуро

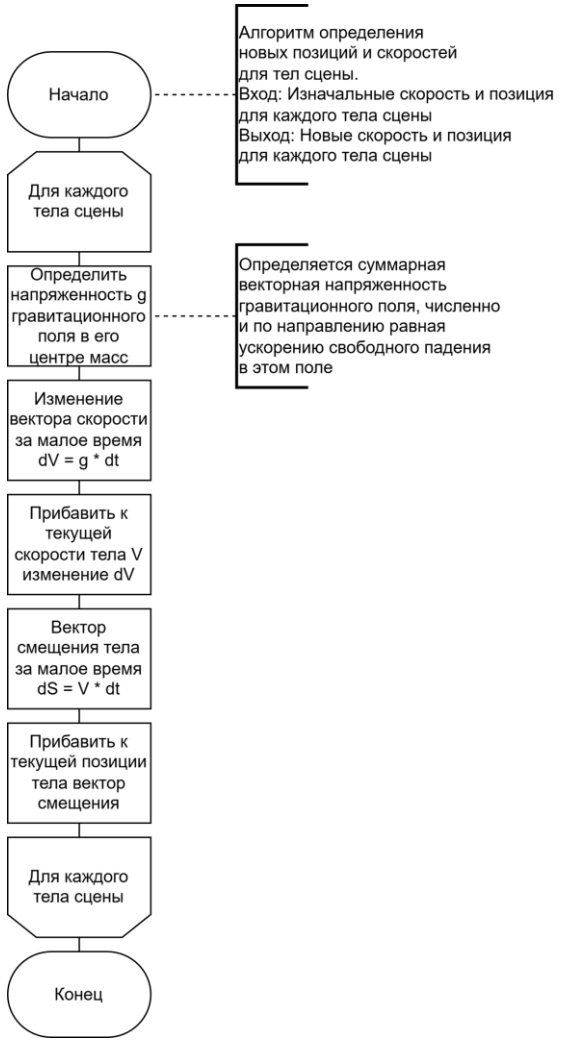
- **Интенсивность вычисляется только в вершинах полигонов и интерполируется внутри полигонов**
- **Можно получить сглаженное изображение с бликами**
- **Меньше вычислений, чем в закраске по Фонгу**

По Фонгу

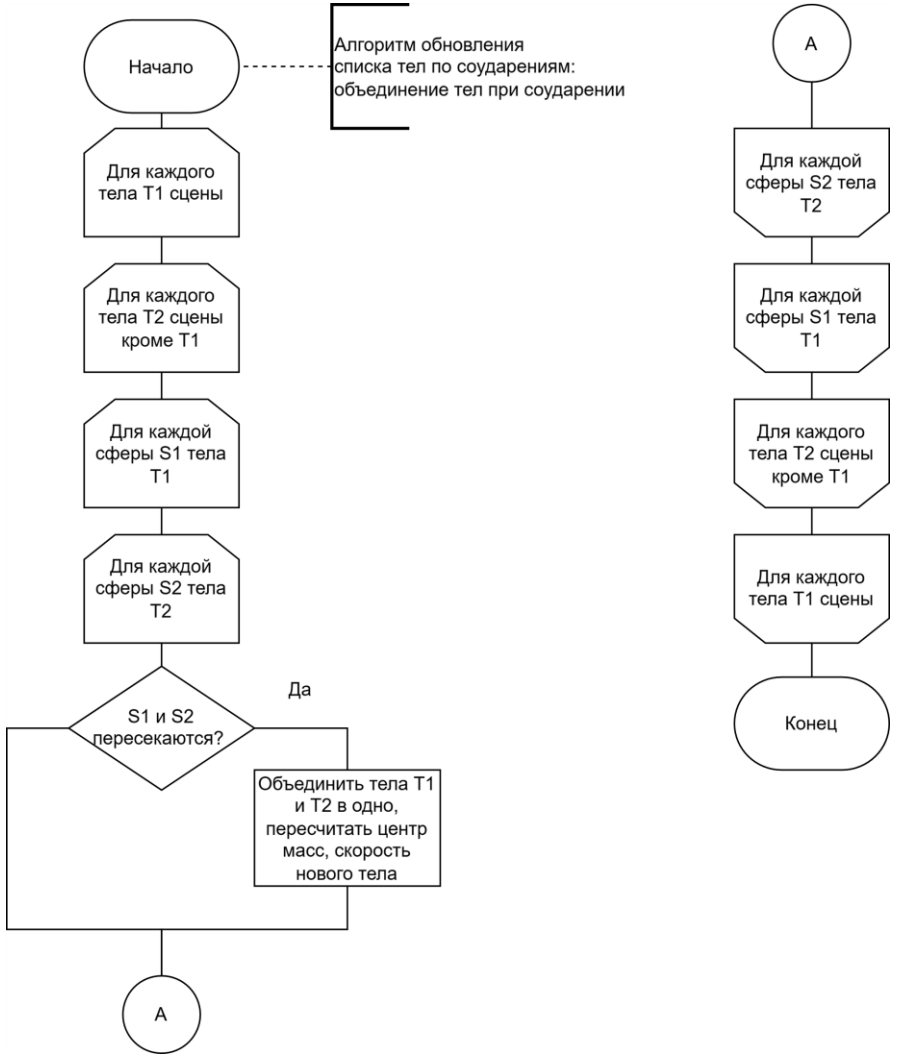
- Вычисляются нормали в вершинах полигонов, далее они интерполируются внутри полигонов
- Можно получить более реалистичные блики, но больше вычислений чем в закраске по Гуро

Закраси однотонная, Гуро, Фонга

Схемы алгоритмов часть 1

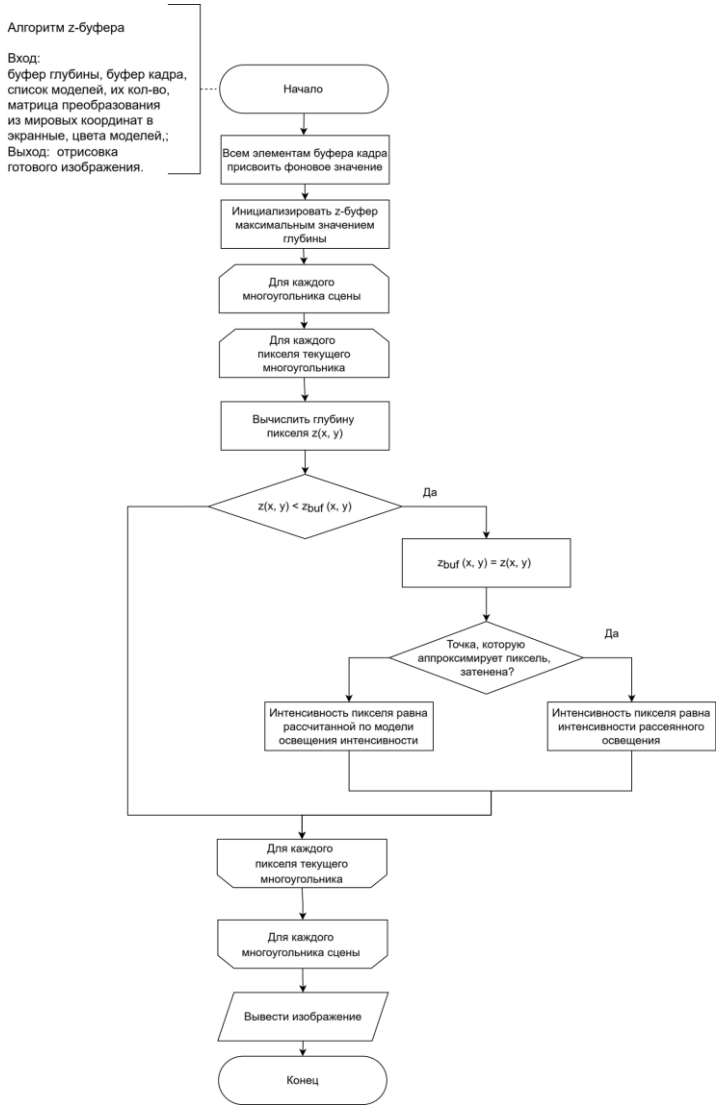


Расчет изменения скоростей и положений тел за δt

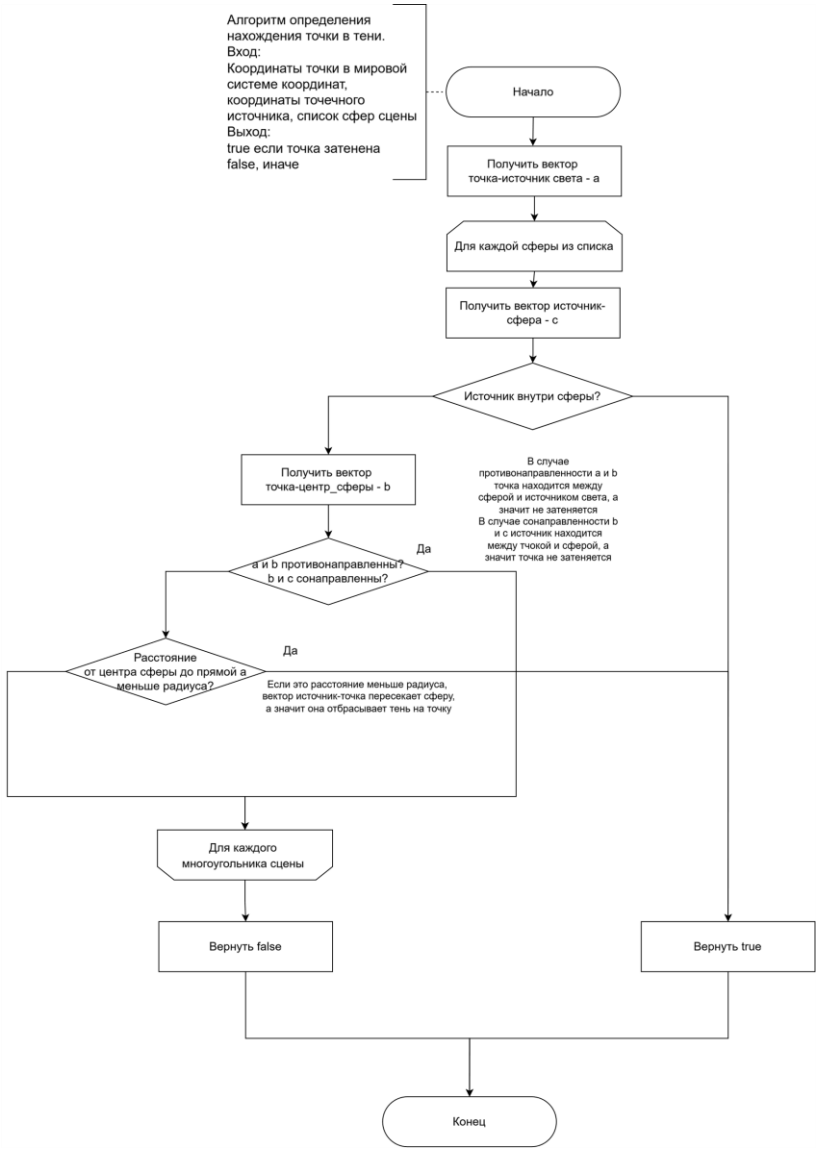


Алгоритм объединения тел при соударениях

Схемы алгоритмов часть 2



Алгоритм Z-буфера



Алгоритм затенения точек

Язык программирования и среда разработки

Был выбран язык C++ и среда разработки QTCteator

- C++ изучался в рамках курса ООП
- C++ является объектно-ориентированным
- QTC позволяет работать с QTDesigner для создания удобного интерфейса пользователя
- QTC обладает всеми средствами создания и отладки ПО

Структура классов

Базовые классы

Object

-vector<shared_ptr<Sphere>>
-Point speed
-Point acceleration
-Point mass_center

+move()
+scale()
+rotate()
+check_intersection(Object)

Sphere

-vector<Point> points
-vector<vector<int>> faces
-Point center
-radius
-mass

+move()
+scale()
+rotate()
+check_intersection(Sphere)

Point

-double x
-double y
-double z

+move()
+scale()
+rotate()
+normalize()

Pixel

-int x
-int y
-double z
-double intensity

Свет и камера

PProjCamera

-Point center
-Point direction
-Point up
-Matrix lookat

+move_forward()
+move_backward()
+move_right()
+move_left()
+move_up()
+move_down()
+rotate_left()
+rotate_right()
+get_projection()

PointLight

-Point center

+move_to(Point)
+move()

Matrix

-vector<vector<double>>
-nrows
-ncols

+identity(nrows,ncols)
+operator* (умножение матриц)
+transpose()
+inverse()
+get_look_at(Point eye, Point center, Point up)

Управление сценой

LoadSolution

-load_solution

+load(file_name)
+load_random()
+set_adapter(adapter)

LoadManager

-load_solution

+load(file_name)
+load_random(n_objs, radius, speed)
+set_adapter(adapter)

Scene

-map<ID, VisibleObject>
-map<ID, InvisibleObject>

+add_object()
+add_camera()
+add_light()
+delete_object()
+delete_camera()
+delete_light()
+calc_accelerations()
+process_collisions()
+sim_iteration()

SceneManager

-scene
-PProjCamera camera
-PointLight light
-DrawAdapter adapter

-set_main_camera()
+set_main_light()
+move_camera()
+move_light()
+sim_iteration()
+draw_scene()
+clear_scene()
+clear_graphics_scene()

Управление отрисовкой

BaseDrawer

-width
-height

+clear()
+add_point()
+draw()
+get_height()
+get_width()

QTDrawer

-width
-height

+clear()
+add_point()
+draw()
+get_height()
+get_width()

DrawerAdapter

-shared_ptr<Object> adaptee
-shared_ptr<BaseCamera> cam
-shared_ptr<BaseLight> light
-vector<vector<double>> zbuf
-vector<vector<double>> intensities
-vector<Sphere> shade_spheres

+draw()
+request()
+set_adaptee()
+set_adaptee_intensities()

QTFactory

+create_drawer()

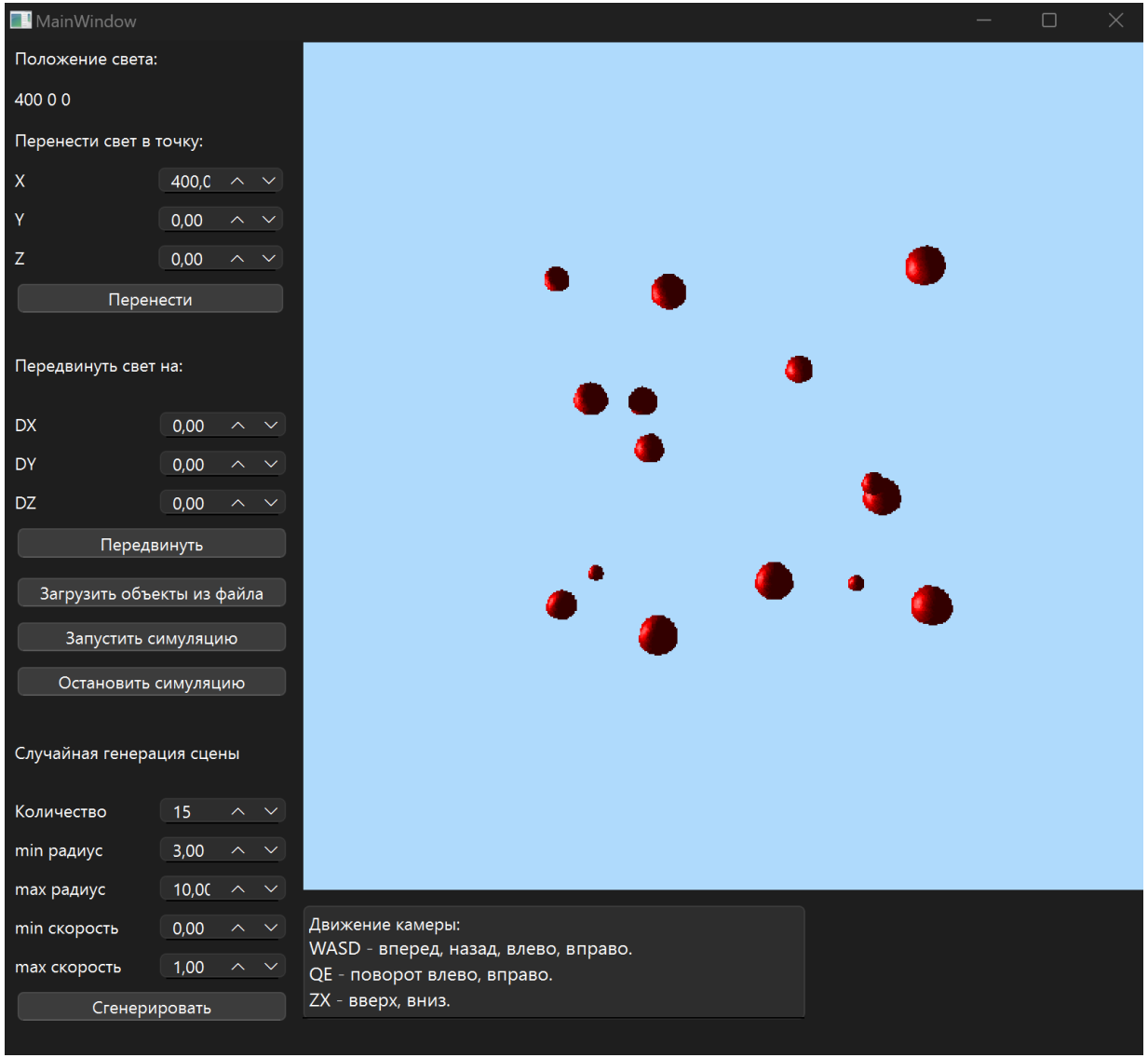
AbstractFactory

+create_drawer()

DrawerSolution

+create_drawer(graphics_scene)

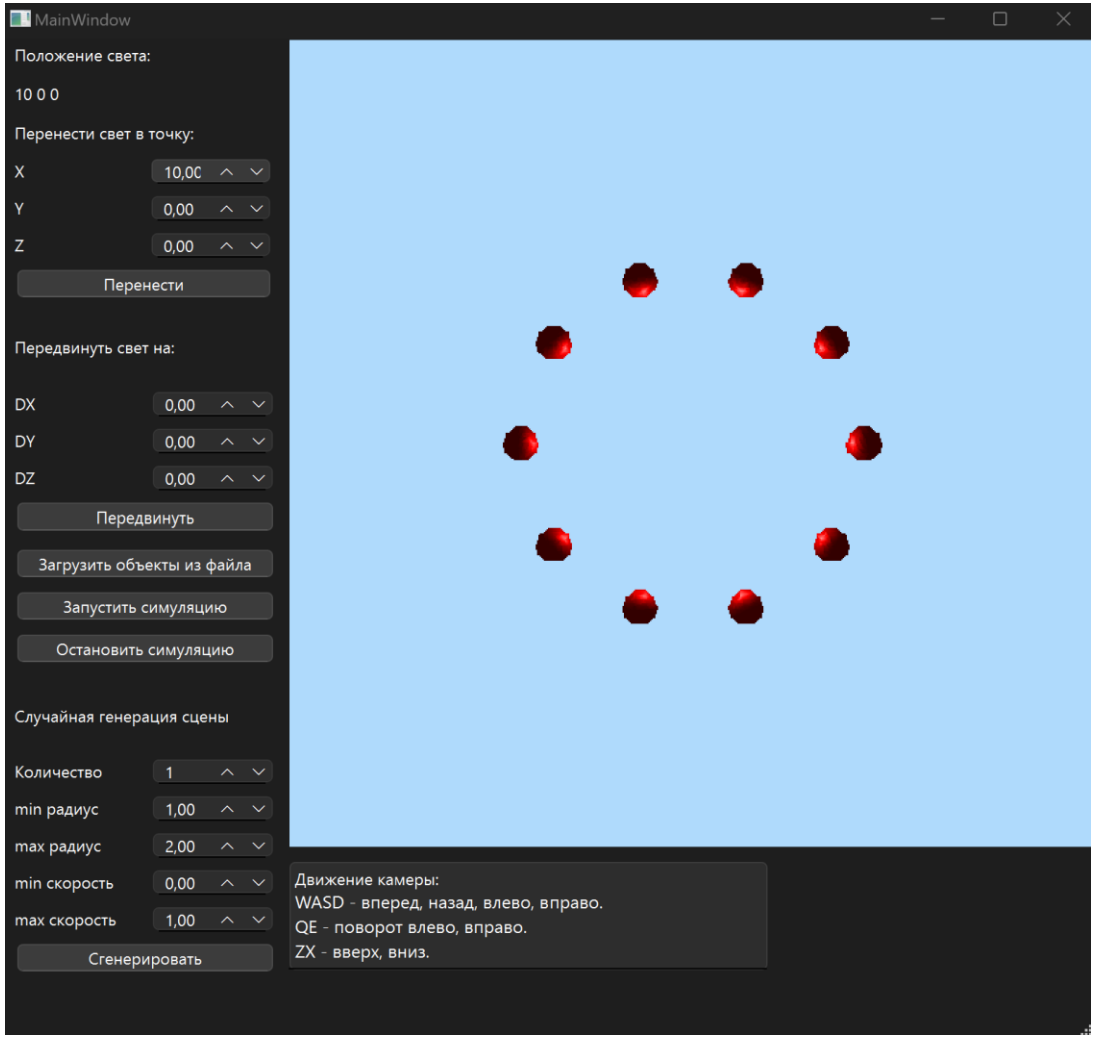
Пример работы и интерфейс



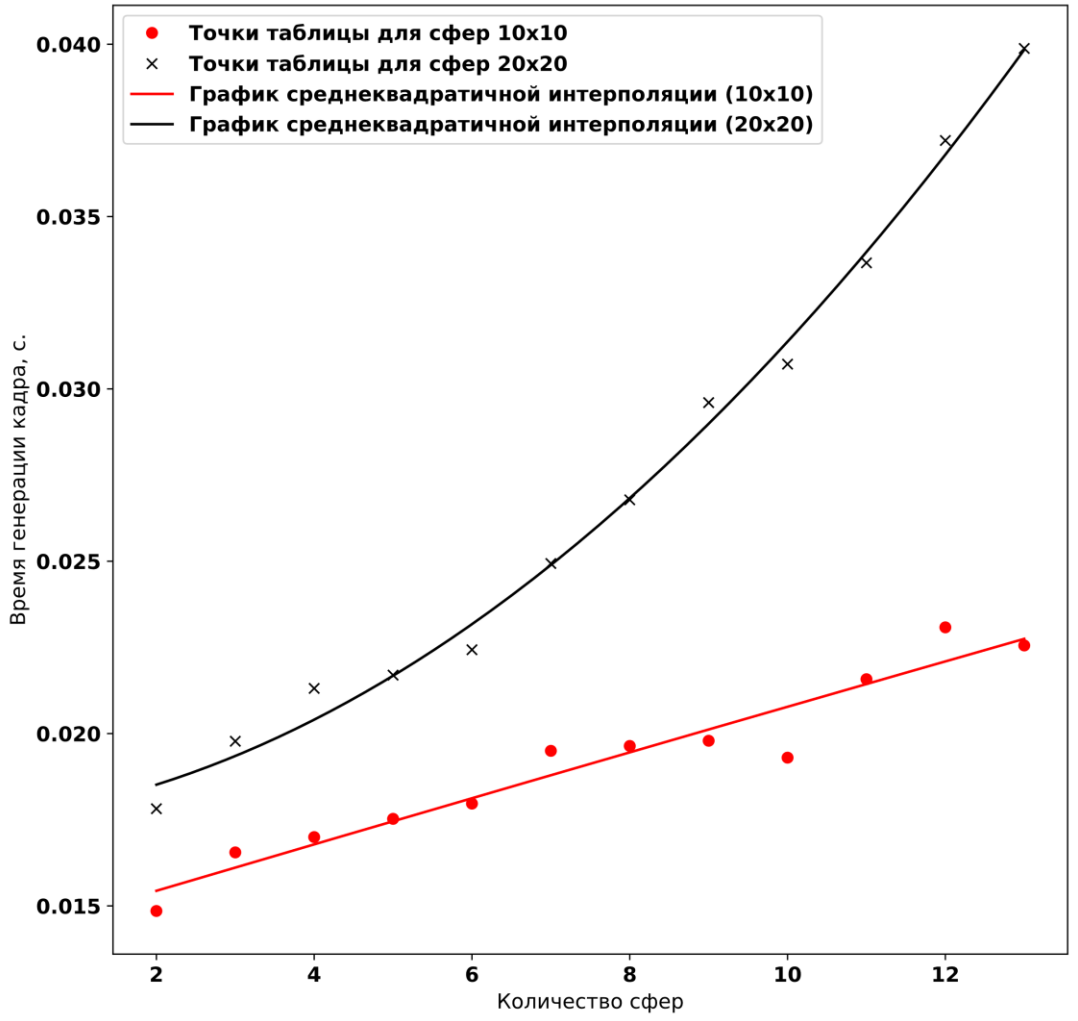
Интерфейс программы позволяет:

- загрузить сцену из правильно сформированного текстового файла
- сгенеририровать случайную сцену по введенным параметрам
- двигать и поворачивать камеру с помощью кнопок клавиатуры
- двигать источник света в указанную точку или на указанный вектор

Исследование



Одна из тестовых сцен



Результаты измерений времени генерации кадра от количества объектов на сцене

Заключение

В рамках курсовой работы были решены следующие задачи:

- Проведен анализ существующих алгоритмов компьютерной графики
- Выбраны подходящие для данной работы алгоритмы
- Выбран язык программирования и среда разработки
- Выбранные алгоритмы реализованы в виде одной программы с графическим интерфейсом
- Проведено исследование быстродействия разработанного ПО
  в зависимости от количества объектов сцены

Поставленная цель достигнута.